

FIG. 1

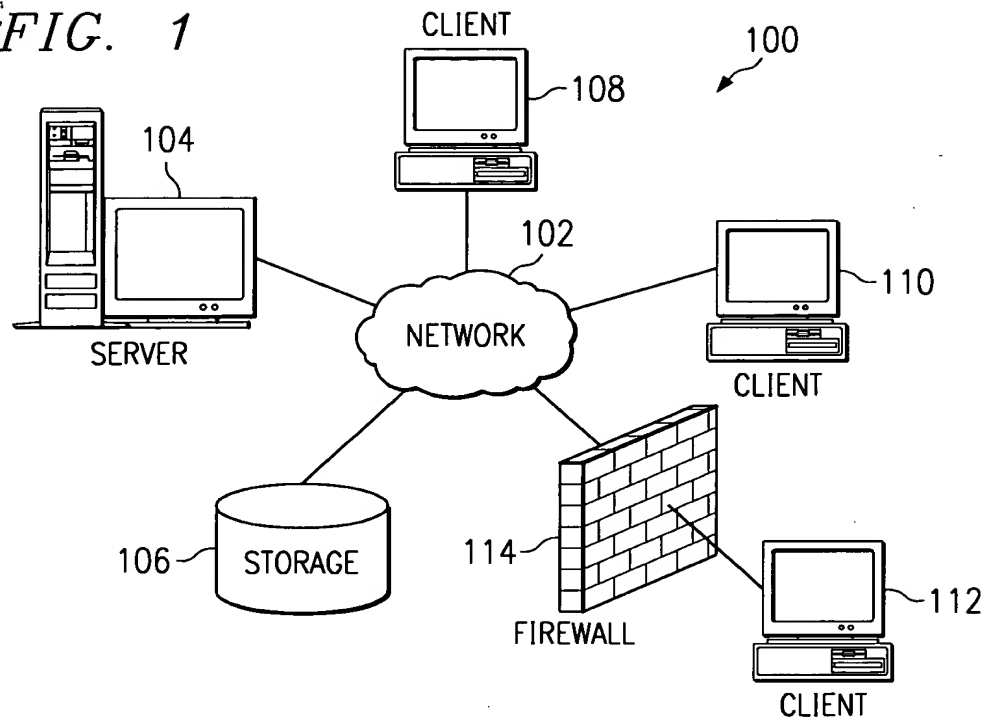
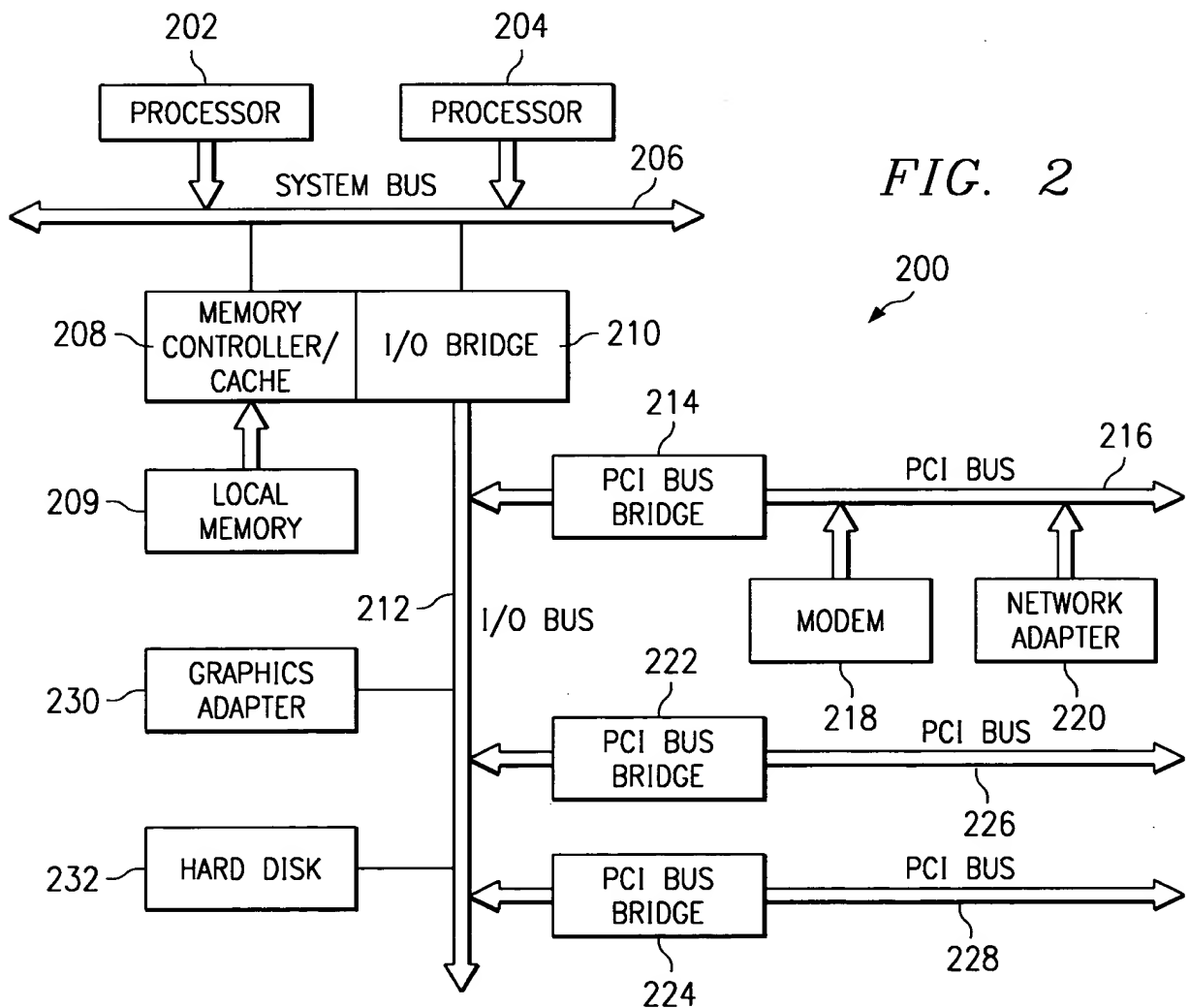


FIG. 2



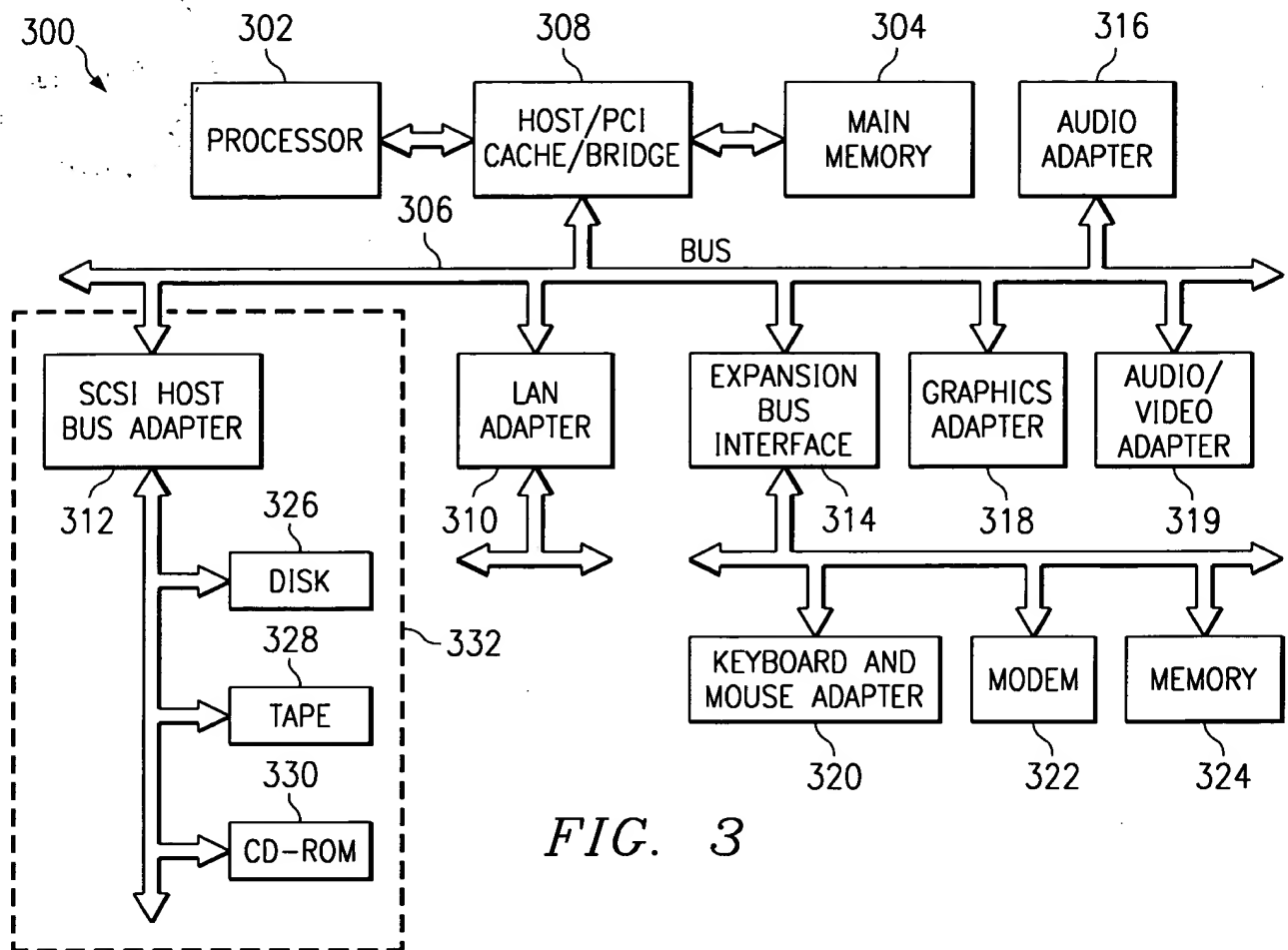


FIG. 3

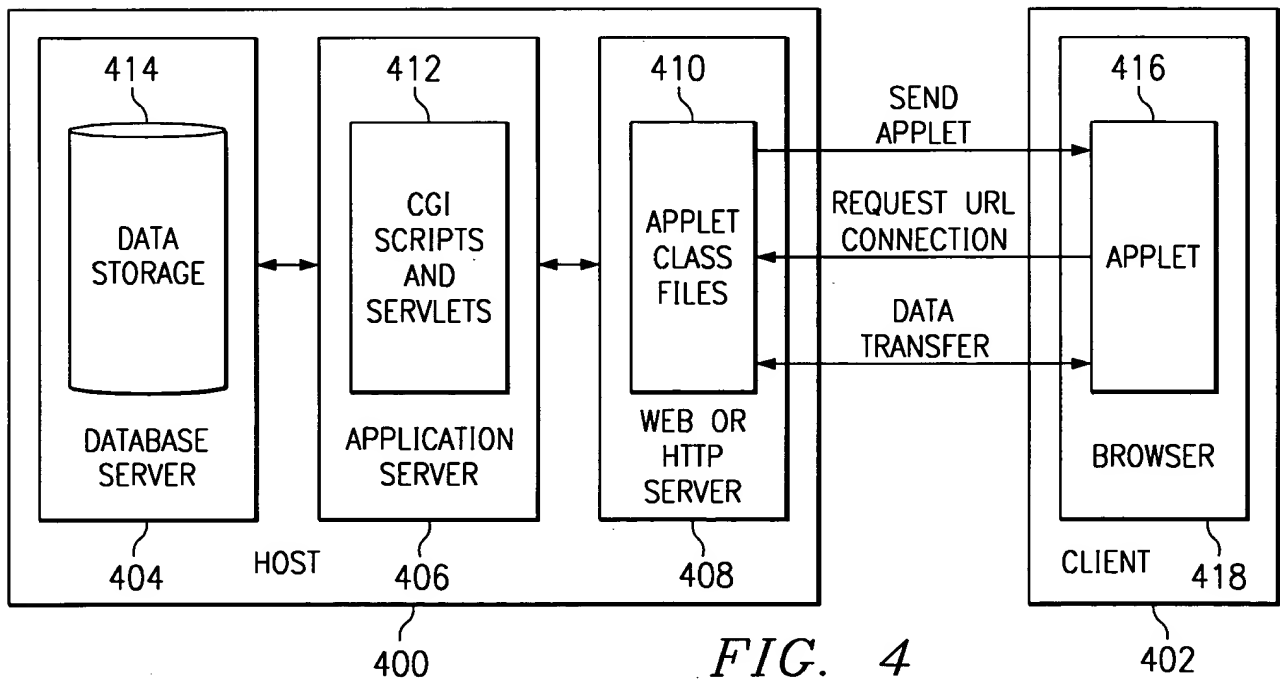


FIG. 4

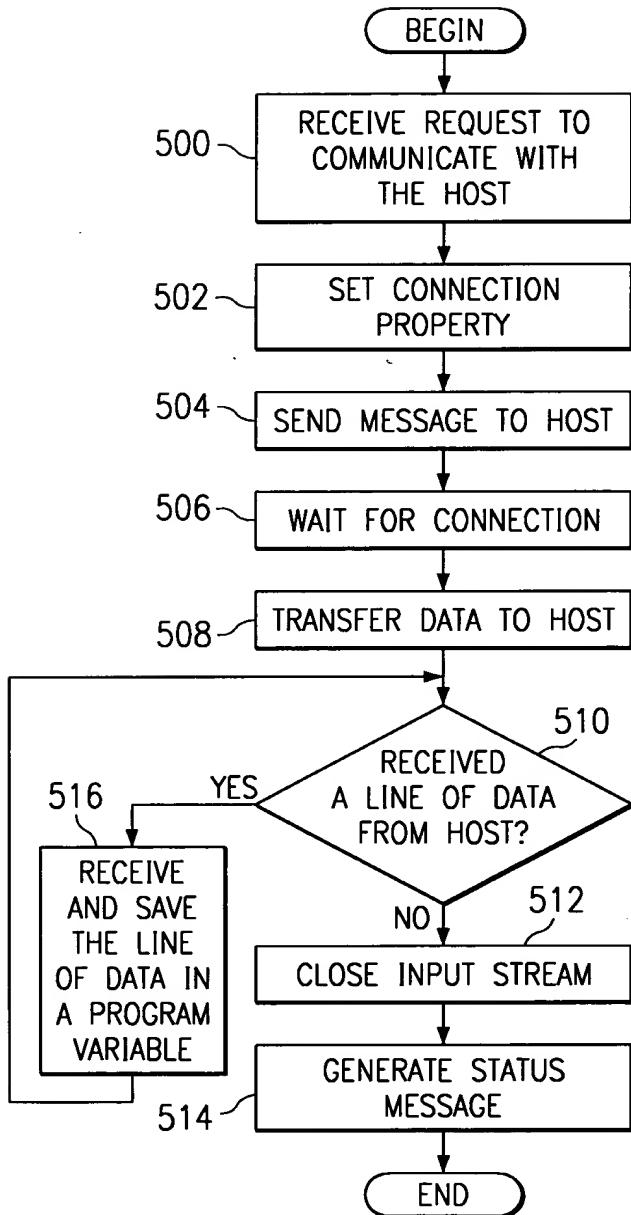


FIG. 5

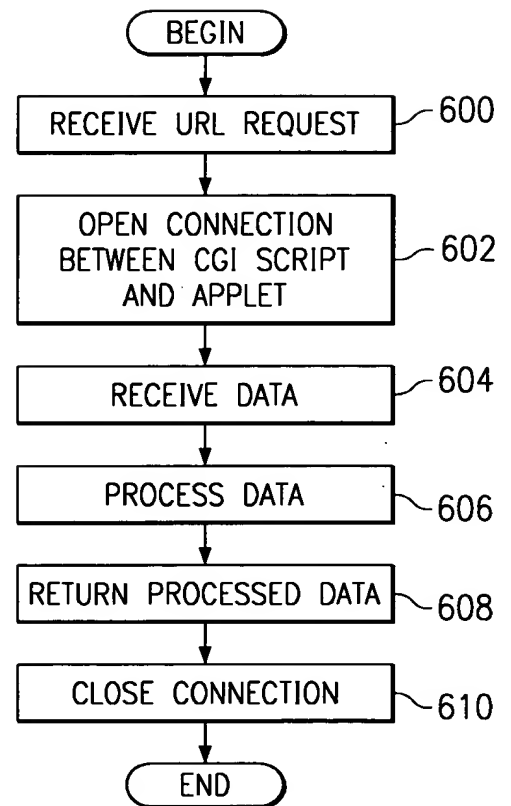


FIG. 6

```

/*
 * This is a function that is executed by the applet. This Java code
 * allows the applet to make a connection and then
 * talk (exchange data) with the host that it was downloaded from.
 *
 * So, create a URL Connection to the server that you (applet) were
 * launched from and then exchange data with the server.
 *
 * Inputs to this function are:
 * 1. aFileAtURL - This is the name of a servlet or CGI program at
 * the server. The applet uses a servlet or CGI
 * helper program back at the host to handle Add,
 * Delete, Update and Retrieve of data from a
 * backend database.
 *
 * 2. dataString - This is the data stream that will be sent to the
 * host once a successful connection is created.

```

FIG. 7A

```

//
public void stdioTalkToHome (String aFileAtURL, String dataString )
{
    // The following 3 vars will be defined before the static {}
    // block in the applet
    URL redirectURL = null;
    static final int DEFAULT_PORT = 80;
    String msgOutString = "";

    URLConnection urlc = null;
    String host;
    int port;
    BufferedReader bri = null;
    DataOutputStream dos = null;

```

700

702

```

// if a msg to the user is already showing, re-surface that msg.
if (getMsgHandle() != null)
{
    getMsgHandle() .changeText("Connecting to host... Please Wait.");
    getMsgHandle() .showit();
}
else
{
    ivMsg = new PopUpMsg( "Applet Status",
        "Connecting to host... Please Wait." );
}

// find the host & port that I (applet) was launched from...
host = getDocumentBase() .getHost();
port = getDocumentBase() .getPort();

consoleDebug("host = " + host);
consoleDebug("port = " + port);

// if port is not set...
if (-1 == port)
{
    // force it to default port...

```

702

704

FIG. 7B

```

// Create the fully qualified URL string...
try
{
    redirectURL = new URL( getDocumentBase().getProtocol(),
                           host,
                           port,
                           aFileAURL );
}
catch (MalformedURLException e)
{
    showHostCommError(1000, "Host URL could not be located for network communication.");
    showError(e);
    return;
}
consoleDebug("Home URL=" + redirectURL);

// Now, create the URLconnection to the host...
try
{
    urlc = redirectURL.openConnection();

    // next, change the setup parameters...
    urlc.setDoInput(true);
    urlc.setDoOutput(true);
    urlc.setUseCaches(false);
    urlc.setAllowUserInteraction(false);

    // and also the request property...
    urlc.setRequestProperty( "Content-Type", "application/x-www-form-urlencoded" );
}

```

706

708

FIG. 7C

```

catch (IOException e)
{
    showHostCommError(1001, "A URLConnection for host communication could not be created.");
    showError(e);
    return;
}
consoleDebug("Home URLConnection was created");

if (getMsgHandle() != null)
{
    getMsgHandle().changeText("Sending data to host... Please Wait.");
    getMsgHandle().showIt();
}

// Send the data from applet to the host server
try
{
    dos = new DataOutputStream(urlc.getOutputStream());
    dos.writeBytes(dataString);
    dos.close();
    consoleDebug("line written: " + dataString);
}
catch (IOException e)
{
    showHostCommError(1005, "User Data write failure occurred during host communication.");
    showError(e);
    return;
}
consoleDebug("All data was sent to host.");

```

708

710

FIG. 7D

```
// Create a Reader
if ( urlc != null )
{
    try
    {
        bri = new BufferedReader(new InputStreamReader(urlc.getInputStream()));
    }
    catch (UnknownServiceException e)
    {
        showHostCommError(1003, "A reading stream for host communication could not be created due to UnknownServiceException.");
        showError(e);
        return;
    }
    catch (IOException e)
    {
        showHostCommError(1003, "A reading stream for host communication could not be created due to IOException.");
        showError(e);
        return;
    }
}
consoleDebug("reader created");

if ( getMsgHandle() != null )
{
    getMsgHandle().changeText("Receiving data from host... Please Wait.");
    getMsgHandle().showIt();
}

// Receive data from host server (using Reader).
String msgOutLineString;
```

FIG. 7E


```
if ( bri != null )
{
    try
    {
        while ( (msgOutLineString = bri.readLine()) != null)
        {
            consoleDebug("msgOutLineString ." + msgOutLineString);

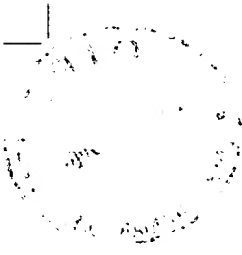
            msgOutString += msgOutLineString + "\n";
        }
    }
    catch (IOException e)
    {
        showHostCommError(1007, "Failure receiving data from host.");
        showError(e);
        return;
    }
}

consoleDebug("msgOutString ." + msgOutString);

consoleDebug("Through reading data from server");

if (getMsgHandle() != null)
{
    getMsgHandle().changeText("Closing host connection... Please Wait.");
    getMsgHandle().showIt();
}
```

FIG. 7F



```
// Cleanup
if ( bri != null )
{
    try
    {
        bri.close();
    }
    catch (IOException e)
    {
        showHostCommError(1008, "Input stream close error during host communication.");
        showError(e);
        return;
    }
}

// Check for and show the user any host communication ERRORS...
.
.
.

// Bring down any communication STATUS msgbox that we were
// showing the user...
.
.
.

} /* end of stdio TalkToHome */
```

714

FIG. 7C

```

/**
 * This function shows debug info in Java Console of the
 * browser only if TRACE flag has been turned on.
 *
 * Inputs to this function are:
 * 1. show - This is the string to display in the Java Console
 *
 */
public static void consoleDebug(String show)
{
    if (TRACE)
    {
        System.out.println("=>" + show);
    }
}
/* end of consoleDebug */

```

714

FIG. 7H